Lecture 9: Predicate Logic

CS 181O
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

First: Finish Propositional Logic in Haskell

Syntax In Haskell

```
data Form =  P String | Ng Form | Cnj [Form] | Dsj [Form]
          deriving Eq

instance Show Form where
 show (P name) = name
 show (Ng  f)  = '-': show f
 show (Cnj fs) = '&': show fs
 show (Dsj fs) = 'v': show fs

form1, form2 :: Form
form1 = Cnj [P p, Ng (P p)]  -- any # args
form2 = Dsj [P p1, P p2, P p3, P p4]
```

From FSynF.hs

Semantics in Haskell

```
-- Find all names in the formula
propNames :: Form -> [String]
propNames (P name) = [name]
propNames (Ng f)   = propNames f
propNames (Cnj fs) = (sort.nub.concat) (map propNames fs)
propNames (Dsj fs) = (sort.nub.concat) (map propNames fs)

-- Generate all valuation for given names
genVals :: [String] -> [[(String,Bool)]]
genVals [] = [[]]
genVals (name:names) = map ((name,True) :) (genVals names)
                ++ map ((name,False):) (genVals names)

-- List of all possible valuations for atoms in formula
allVals :: Form -> [[(String,Bool)]]
allVals = genVals . propNames
```

Semantics in Haskell

```
-- eval takes valuation and formula and gives value
eval :: [(String,Bool)] -> Form -> Bool
eval [] (P c)    = error (no info about  ++ show c)
eval ((i,b):xs) (P c)
    | c == i        = b
    | otherwise = eval xs (P c)

eval xs (Ng f)   = not (eval xs f)
eval xs (Cnj fs) = all (eval xs) fs
eval xs (Dsj fs) = any (eval xs) fs
```

From FSemF.hs *line 64*

Semantics in Haskell

```
- - Is formula a tautology or satisfiable or a contradiction
tautology :: Form -> Bool
tautology f = all (\ v -> eval v f) (allVals f)

satisfiable :: Form -> Bool
satisfiable f = any (\ v -> eval v f) (allVals f)

contradiction :: Form -> Bool
contradiction = not . satisfiable

- - Does first formula logically imply second
implies :: Form -> Form -> Bool
implies f1 f2 = contradiction (Cnj [f1,Ng f2])

-- If start with list of vals and formula F then returns sublist making F true
update :: [[(String,Bool)]] -> Form -> [[(String,Bool)]]
update vals f = [ v | v <- vals, eval v f ]
```

Predicate Logic

Syntax

- Symbols needed include
  - variables: x, y, ...
  - constant symbols: c, d, ...
  - k-ary function symbols: f^k, g^k, ... for all k ← *return values*
  - k-ary predicate symbols: P^k, Q^k, ... for all k ← *are true or false*
  - parentheses: (,)
  - quantifiers: ∃, ∀
  - logical connectives: ¬, ∧, ∨, →

Terms & Formulas

- Atomic formulas are built by applying relation symbols to variables:

  v ::= x | y | z | **v**'

  c ::= c | d | **c**'

  f ::= f | g | **f**'

  t ::= **c** | **v** | **f tlist**

  tlist ::= [] | **t**: **tlist**

  R ::= P | R | S | **R**'

  atom ::= **R tlist**

Terms & Formulas

- Formulas:

F ::= **atom** | (**t** = **t**) | (¬**F**) | (**F**∨**F**) | (**F**∧**F**) | (∀**v**. **F**) | (∃**v**. **F**)

Examples

- Let D(x) stand for x is a dog, B(x,y) for x bites y, P(y) for y is a person, s for Sally, and f for Fido
- Fido bit someone
  - ∃*x*.(P(x) ∧B(f, x))
- Every dog bit Sally
  - ∀x.(D(x) → B(x,s))
- Some dog bit Sally
  - ∃x.(D(x) ∧ B(x,s))

Examples

- Let L(x,y) stand for x loves y.
- Everybody loves somebody
  - ∀x.(P(x) → ∃y.(P(y) ∧ L(x,y)))
- Someone loves everyone
  - Ambiguous: ∃x.(P(x) ∧ ∀y.(P(y) → L(x,y))) *or* ∀y.(P(y) → ∃x.(P(x) ∧ L(x,y)))
- Jane's mother loves her
  - L(mother(Jane),Jane) where mother() is a unary function

Limiting Domains

- Every person hates a wall
  - $\forall x. (P(x) \rightarrow \exists y. H(x,y) \wedge W(y))$
  - $\forall x. \forall y.(P(x) \wedge W(y) \rightarrow H(x,y))$????
- There is a wall that is hated by all people.
  - $\exists y. W(y) \wedge \forall x. (P(x) \rightarrow H(x,y))$

Free & Bound Variables

- Historically confusing! (But like lambda calculus)
- In Love(x,y) the variables x and y are free
  - the meaning of the wff depends on the meaning of x,y
- In $\forall x.\exists y.L(x,y)$ occurrences of x and y are bound by the quantifiers.
  - Meaning does not depend on meanings of x, y.

Free Variables

- An occurrence of x in ϕ is free in ϕ if it is a leaf node in the parse tree of ϕ such that there is no path upwards from that node x to a node $\forall x$ or $\exists x$.
- Otherwise, that occurrence of x is called bound.
- For $\forall x\phi$, or $\exists x\phi$, we say that ϕ – minus any of ϕ's subformulas $\exists x\, \psi$, or $\forall x\, \psi$ – is the scope of $\forall x$, respectively $\exists x$.

Examples

- $(\forall x.\exists y.L(x,y)) \wedge H(x,y)$
  - Some occurrences free and some bound.

Substitution

- Define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t.
  - Expect $\forall x.\phi(x) \Rightarrow \phi[t/x]$ for every term t
  - What about $\forall x.\exists y.L(x,y) \Rightarrow \exists y.L(y,y)$?

More Substitution

- Say that *t is free for x in ϕ* if no free x leaf in ϕ occurs in the scope of $\forall y$ or $\exists y$ for any variable y occurring in t.
  - y not free for x in $\exists y\ L(x,y)$
  - Only allow substitution $\phi[t/x]$ if t free for x in ϕ
  - If t not free for x in ϕ, rename bound variables to make substitution legal.

Typed Predicate Calculus

- Variant where bound variables have types
  - $\exists x$: T, $\forall y$: U
- Examples:
  - Fido bit someone $\Rightarrow$ $\exists x$: Person. B(f, x))
  - Every dog bites Sally $\Rightarrow$ $\forall x$: Dog. B(x,s))
  - Some dog bit Sally $\Rightarrow$ $\exists x$: Dog. B(x,s))
- Can be translated away

Questions?